Exploring Model Building for Natural Language Understanding

Johan Bos

Institute for Communicating and Collaborative Systems
Division of Informatics, University of Edinburgh
2 Buccleuch Place, Edinburgh EH8 9LW, Scotland, United Kingdom
jbos@inf.ed.ac.uk

Abstract

In this paper the use of model building for natural language understanding in practical systems is investigated. After outlining several interpretation tasks where model construction would be useful, we perform experiments using two state-of-the-art model builders on the interpretation of imperatives in discourse. The results are acceptable for small discourses and small domains, but don't scale up for larger domains or longer discouses: the complexity of model generation rises sharply in relation to the difficulty of a task and the size of the dialogue. Several suggestions are provided for future research to improve the performance of model building for larger domains and richer texts.

1 Introduction

This paper introduces a discipline in the field of automated reasoning that has received only little attention within computational semantics: *model building* (sometimes also referred to as *model generation*, *model construction*, or *model searching*) for first-order logic. Automatic model builders offer a positive handle on the satisfiability problem and are therefore often used in tandem with theorem provers (who offer a negative handle on satisfiability). But a model builder, as its name suggests, has another attractive property: it is able to construct concrete models for first-order theories. As I will show in this paper, the representations of such models are not only flat and extremely easy to process—they also embody the information required for many natural language understanding tasks.

Although it is certainly true that automated model building hasn't reached a state of matureness that automated theorem proving has achieved in the last decades, it is also fair to say that performances in model building have significantly improved the last years and have reached a level close to be useful in linguistic applications. These recent developments haven't gone entirely

unnoticed by computational linguists—as recent applications for question answering [11,2] and natural language disambiguation [6,7] show. Nevertheless, I would like to position model building as a central task in natural language understanding. In particular, model building might play an important role in linking up natural language processing front ends with information or actions anchored in databases or situations.

Let's be more concrete about this and exemplify what roles theorem proving and model building might play within a natural language understanding system. Let's assume this system captures the discourse in a semantic representation (i.e., the logical form) compatible with first-order logic. Given a portion of discourse D and a goal G, the proposed procedure runs along the following steps:

(i) *Construct a set of first-order representations* Θ *for* D.

(ii) *For each* $\theta \in \Theta$*, attempt to build a model for* θ *and background knowledge* Φ*, by simultaneously:*
   (a) *giving* $(\theta \wedge \Phi)$ *to a model builder, possibly resulting in a model* M *(for consistent theories* θ*).*
   (b) *giving* $\neg(\theta \wedge \Phi)$ *to a theorem prover, possible resulting in a proof (for inconsistent theories* θ*). Remove* θ *from* Θ.

(iii) *Use* M *to extract information required for* G *using a model checker.*

Step (i) presupposes tools that construct semantic representations for natural language discourses. Due to ambiguities inherent in natural language, this results usually in a set of logical forms. This process can involve speech recognition, parsing, semantic construction, and ambiguity resolution. Various of these tools are available nowadays [2], albeit for small domains and restricted coverage. Finally, note that semantic representations need not be in directly in first-order format. For instance, translations from Discourse Representation Structures [8] to first-order logic are available [3].

Step (ii) attempts to construct a model for theory θ. The result of ambiguity resolution as part of the previous step of processing might result in several alternative theories, some of them being inconsistent. Now, theorem provers are used to detect inconsistent theories, and model builders to find minimal models for consistent theories. The concept of minimality is important here, because it ensures that no redundant information will enter M. For efficiency reasons, this step should ideally be realized by running (ii)(a) and (ii)(b) in parallel.

Step (iii), finally, links models to actions. That is, we use the information provided by the models to carry out certain tasks, such as retrieving a database or letting a robot perform some physical action. Once you know what you're looking for in the model, given a goal G, this is the easiest part of the procedure. It is also convenient to use a model checker for this task, as illustrated in [2]. I will illustrate the use of model checkers in Section 2.5.

As will be clear from the procedure outlined above, this framework for natural language understanding uses an explicit intermediate level of representation, namely the model constructed in Step (ii). There are two (related) reasons that motivate this additional layer in the architecture:

- models are flat structures and hence extremely easy to process
- models contain no explicit quantification or boolean structures

This is precisely why model building is a useful tool for applications in computational linguistics, particularly in combination with model checking and theorem proving. Model building offers us simple means to construct normalised information states from semantic representations, performing all the necessary reasoning involving quantification and boolean connectives. In addition, model checking techniques allow us to extract information from constructed models or test for certain properties of models. Theorem proving will help us to detect inconsistent theories.

In the remainder of this paper, I will underline and further exemplify these points by first presenting the structure of first-order models, and show how they are constructed for first-order theories (Section 2). Specific applications, from a computational linguistic point of view, are presented and demonstrated in Section 3. I will show the practical use of model building by presenting results for two state-of-the-art model builders for first-order logic in Section 4.

2 Model Building

2.1 Off-the-Shelf Model Builders

Simply put, a model builder is a system that tries to generate a finite model for a given theory (a set of first-order formulas). This is essentially done by systematically searching in the space of possible models, given by universes and interpretations. Throughout the paper we will discuss practical experience obtained from two model builders:

- MACE 2.0 (developed by Bill McCune)
  `http://www-unix.mcs.anl.gov/AR/mace/`
- PARADOX 0.4.1 (developed by Koen Claessen and Niklas Sörensson)
  `http://www.math.chalmers.se/~koen/paradox/`

MACE uses the Davis-Putnam-Loveland-Logeman decision procedure to find models. The input of MACE can be formulated in first-order formula syntax. MACE is an excellent inference tool—performing a tandem with the theorem prover OTTER it was winner in the SAT division (a collection of first-order satisfiable problems) of the CASC-16 competition. [1] PARADOX is

[1] CASC is short for the CADE ATP System Competition, CADE is the annual international Conference on Automated Deduction. Information on the latest CADE can be found on `http://www.CADE-19.info/`.

a newly developed tool using a similar algorithm as MACE but with further optimisations. PARADOX was winner of the SAT/Model class at CASC-19 and, as we will see in Section 4, it outperforms MACE for the linguistic problems we consider.

In the scope of this paper we will only consider the performances of MACE and PARADOX, to my knowledge (and experience) two of the best model builders for first-order logic around today. Other model builders that I expect will perform well on linguistic problems are GANDALF (developed by Tanel Tammet) and MACE4 (also known as ICGNS).

2.2 First-Order Models

Before introducing the actual process of model building, let us recall the nature of first-order models. In a fairly abstract way, models can best be viewed as a description of certain aspects of a situation. First-order models, to be more precise, consist of two parts: a non-empty domain D of objects (or *individuals*) and an interpretation function F that assigns properties to these objects. Models are always given with respect to a certain *signature*. A signature tells us which symbols we use in our semantic representations, and what arity they have. For instance, here is a signature:

0-place predicates: `mia, vincent, butch`
1-place predicates: `man, woman`
2-place predicates: `know`

A signature (or a vocabulary) is just there to keep track of the symbols we use, and how we use them. F will assign values following the specification in the signature: it will map 0-place predicates (or *constants*) to members of D, it will map 1-place predicates to sets of members of D, and it will map 2-place predicates to sets of pairs of members of D. To give an example, let's consider a model over the above signature for a situation snap shot starring three individuals: a woman named Mia, and two men named Vincent and Butch. In this situation Butch knows Vincent, and Vincent knows both Mia and Butch:

```
D={d1,d2,d3}

F(mia)=d1           F(man)={d2,d3}
F(butch)=d2         F(woman)={d1}
F(vincent)=d3       F(know)={(d2,d3),(d3,d1),(d3,d2)}
```

The models we work with in the scope of this article are taken to be implicitly representing negative information (cf.[2]). So from the above model we can conclude that Mia is not a man, that Vincent and Butch aren't women, that Mia does not know any of the other persons in the domain, and so on. In fact, according to this model, none of the persons knows him/herself.

Suppose we work with the signature introduced above. Now let's construct a first-order theory (a set of formulas) describing the situation where Mia does not know Vincent, every man knows a woman, Butch is a man, Vincent is a man, Mia is a woman, and men are disjoint from women. These statements can be represented by the following first-order theory:

$\neg$know(mia,vincent) $\quad$ $\forall$x(man(x)$\rightarrow$ $\exists$y(woman(y)$\wedge$know(x,y)))
man(butch) $\quad$ man(vincent) $\quad$ woman(mia) $\quad$ $\forall$x(man(x)$\rightarrow$ $\neg$woman(x))

To check whether this theory is consistent (or *satisfiable*), we try to find at least one model that satisfies all the formulas of the theory. A model builder systematically attempts this by first considering a model with a domain of a single individual, and if it fails to do so it will try to check whether there is a model with a domain consisting out of two members—and so on—until it finds one. Both MACE and PARADOX use this process of iteration, although this procedure can be sped up by specifying an expected domain size or range of domain sizes by an additional command-line parameter.

For the rather simple theory above, a model builder will soon find out that it is impossible to construct a model that contains just one individual. For, according to our theory, Vincent is a man and Mia is a woman, and men are disjoint from women. However, the model builder will succeed in finding a model with a domain of two elements, namely the following one:

```
D={d1,d2} F(mia)=d1          F(man)={d2}
          F(butch)=d2        F(woman)={d1}
          F(vincent)=d2      F(know)={(d2,d1)}
```

Note that this model might not be intuitive at first sight—it assigns the same individual (d2) to both Vincent and Butch. As model builders attempt to find *minimal* models, and because we didn't say anything about Butch and Vincent being two different persons, this is a proper model for our theory. (So incidentally, model building also turns out to be a valuable method for verifying the correctness of background knowledge—models often reveal insights that were unclear from the difficult to read axioms that define the knowledge that support semantic analysis.) We might revise our theory, by adding extra information such as $\neg$butch=vincent. This will yield models with at least three entities. Alternatively, we might consider only so-called *Herbrand models*, which is a class of models where all 0-place predicates (the constants) map to themselves in the model's domain. Indeed, the model builder MACE comes with features to simulate properties of Herbrand models, but we won't explore this feature in this paper.

A little bit more needs to be said about minimal models. The concept of minimality can be applied to (i) the number of entities in the model's domain, and (ii) the number of positive predicate extensions (for predicates with an arity higher than 0). Both MACE and PARADOX will generate models with

a minimal domain by default. PARADOX however, using heuristics for initial settings for predicate extensions, does not necessarily produce models with minimal predicate extensions.

Going back to the example, it is important to note that there are infinitely many models satisfying the input theory, and in general there could be several minimal models output by the model builder. However, there is also the possibility that the model builder won't succeed in finding a model, even though the input theory is consistent. Consider the following theory:

$$\texttt{person(butch)} \qquad \forall \texttt{x(person(x)} \rightarrow \exists \texttt{y(person(y)} \wedge \texttt{parent(x,y)))}$$
$$\forall \texttt{x} \forall \texttt{y} \forall \texttt{z(parent(x,y)} \wedge \texttt{parent(y,z)} \rightarrow \texttt{parent(x,z))} \qquad \neg \exists \texttt{x parent(x,x)}$$

The model builders that we are considering here are only able to construct *finite* models, that is models with a fixed domain-size. The above theory has no finite model: for each person has a parent, which is also (a different) person, and that person has a parent, which is also a person, and so on. Now, although the problem of finding a finite model for a given domain size is decidable, most model builders, including MACE and PARADOX, require a range of domain sizes in which to search for models in order to avoid getting entangled in an infinite loop.

For the above input, both MACE and PARADOX will find out relatively quickly (depending on the estimated domain size) that there are no models for the given input theory. Estimating the expected domain size is a non-trivial matter but important for high-speed performances, as will be shown in Section 4. One way to derive model-size estimations in practical systems is using machine learning techniques—a method which needs to be explored in future work.

2.4 Inconsistent Theories and Theorem Proving

It is easy to come up with a theory that has no model. For instance, we could describe the unlikely situation where Butch happens to be a man and a woman, and where men are disjoint from women:

$$\texttt{man(butch)} \qquad \texttt{woman(butch)} \qquad \forall \texttt{x(man(x)} \rightarrow \neg \texttt{woman(x))}$$

It is impossible to construct a model such that all of the above formulas are true. The theory is *inconsistent* or *unsatisfiable*. In general it is not advisable to use model builders to detect inconsistencies. For extremely simple theories, such as the above, model builders will be able to do so, but when theories are more elaborate they will reach a stage of computational unpleasantness.

Theorem provers are precisely the tools one needs to detect inconsistencies. This is done by giving the negation of the theory to a theorem prover. If a proof is found (the negated theory is a validity, i.e. true in any model), this implies the original theory is unsatisfiable. Some state-of-the-art theorem provers suitable for this task are:

- BLIKSEM

http://www.mpi-sb.mpg.de/~bliksem/

- SPASS
  http://spass.mpi-sb.mpg.de/
- VAMPIRE
  http://www.cs.man.ac.uk/~riazanoa/Vampire/

The problem of detecting whether a first-order formula is valid is not decidable. In theory, this means that a theorem prover might never come back with an answer for certain input formulas. In practice, this means that one needs to specify certain amounts of resources (time and memory allocations) that theorem provers can use when attempting to find a proof. In a setting where theorem provers and model builders are used to work on the same problem, they can help each other out. If the model builder finds a model for ϕ while a theorem prover is attempting to find a proof for $\neg\phi$, the theorem prover can be informed to quit attempting to prove the conjecture, because it will never succeed in doing so. Similarly, if the theorem prover proves $\neg\phi$, the model builder will never be able to find a model for ϕ.

2.5 Model Checking

A model builder is a tool that, given a first-order theory, produces a finite model if there exists one. A *model checker*, on the other hand, is a tool that given a model and a first-order query, tells us whether that model satisfies the query or not. The simplest form of queries are first-order sentences (formulas containing no free variables), prompting the model checker to return yes or no (assuming the query is built over the same vocabulary of symbols as the model). More complex queries can contain free variables, which in the process of model checking will be bound to specific values of the model's domain. An example of a model checker that allows the latter option is provided in [2].

3 Linguistic Applications

Model building deals automatically with quantification and boolean operators. The property of constructing *minimal* models, and the accessible structure of the generated models themselves, make model builders interesting to use in linguistic applications. Minimality ensures that no redundant or irrelevant information will be specified in the model. The flat structures ensure whatever information is searched for can be accessed without any appeal to recursion or forms of heavy inference. Furthermore, model checkers allow one to query generic properties formulated in first-order logic.

We will exemplify the idea of using first-order models with examples from natural language understanding systems, involving question answering and action interpretation. A system that implements these ideas consists at least of a natural language component that maps natural language into semantic representations, a model builder, a theorem prover, and (optionally) a model

checker. The semantic representation captures the ongoing dialogue between user and system, and forms the input to the model builder (to generate a model) and the theorem prover (to check for contradictions). The generated model is the primary information source for the system to decide what question to ask next or what action to perform.

3.1 Information Seeking Dialogue

The situation considered here is one where the system provides the caller a route description on the basis of the caller's destination, origin of travel, time, and whether he or she wants the quickest or the shortest route. Traditionally this is done via *slot filling*, where a frame suitable to a specific application (here route descriptions) comes with certain slots (here: `destination`, `origin`, `time`, `kind-of-route`) and possible values. In the course of the dialogue, these slots gets instantiated, and as soon as all slots are filled the system will calculate a route and present it to the user. We won't abandon this method as such—what we will propose to change though is *how* these slots are filled. In fact what we show is that models are in a way very general (and flexible) frames.

An example dialogue in this domain could run as follows.

S: Where do you want to go to? (1)
U: I would like to go from Edinburgh to London. (2)
S: When would you like to go from Edinburgh to London? (3)
U: Tomorrow. (4)
S: OK. Please wait while I calculate your route. (5)

The semantic representation for utterance (2) is (ignoring temporal and modal information for clarity):

$\exists$`e(go(e)`$\wedge$`agent(e,user)`$\wedge$`from(e,edinburgh)`$\wedge$`to(e,london))`

The next step is to generate a model for this representation. As remarked before we normally provide background knowledge in the form of first-order axioms to construct the models we desire. The relevant background knowledge for the verb "to go" can be stated as follows:

$\forall$`e`$\forall$`x`$\forall$`a(go(e)`$\wedge$`agent(e,a)`$\wedge$`from(e,x)`$\rightarrow$`origin(a,x))`
$\forall$`e`$\forall$`x`$\forall$`a(go(e)`$\wedge$`agent(e,a)`$\wedge$`to(e,x)`$\rightarrow$`destination(a,x))`
$\forall$`e`$\forall$`x`$\forall$`a(go(e)`$\wedge$`agent(e,a)`$\wedge$`temploc(e,x)`$\rightarrow$`time(a,x))`

These formulas say something about the relationship between the event translated to the first-place relation symbol `go` and its argument structure provided in a neo-Davidsonian style and the slots `origin` and `destination`, both represented as two-place relations. (In general there are many of these axioms required for a system to be interesting, and further ontological knowledge is required—not shown here—that supplies information about disjointness and inheritance of concepts.) Now consider the model generated for the semantic

representation for (2) supported by the background knowledge axioms given above:

```
D={d1,d2,d3,d4}

F(user)=d1          F(agent)={(d2,d1)}     F(origin)={(d1,d3)}
F(go)={d2}          F(from)={(d2,d3)}      F(time)={}
F(edinburgh)=d3     F(to)={(d2,d3)}
F(london)=d4        F(destination)={(d1,d4)}
```

The system's next response is solely based on the content of this model. The next utterance of the system (3) is prompted because `time` is the only "slot" in the model that hasn't got a value yet.

Surely, any system without a deep semantic analysis can get similar results by using key word spotting techniques and similar patterns as described in our background knowledge to find the appropriate values for the slots of the application. But there is a crucial difference. The model building approach naturally deals with negation and disjunction. Consider the following more complex examples of potential user utterances:

I want to fly from Heathrow or Gatwick. (6)
I do not want to fly to Heathrow. (7)
I'd like to fly from any London airport except from Luton on Mondays. (8)

In (6) the speaker does not commit him/herself to the starting location of travelling. The model building approach will generate two minimal models for this case. (7) contains negated information. Here, the generated model will never associate Heathrow with the slot for destination. Even more complicated reasoning is involved in (8), which a model builder will perform automatically.

3.2 Controlling a Mobile Robot

In this scenario we consider an indoor mobile robot which can be controlled by natural language instructions, and show how model building is useful to determine what action the robot should perform in a given situation. Directives can express simple or complex commands, but can be warnings or advice as well, as is exemplified by the following instructions given by the user to the robot:

Go to the kitchen! (9)
Clean all the rooms on the second floor! (10)
Go forward and you will fall down the stairs. (11)

Again we translate natural language statements into a semantic representation. To deal with actions such as above in (9), we will adopt a slightly more complicated version of the translation into first-order logic. Actions are represented as three-place relation between a situation that holds the information before the action is performed, a situation that describes the action, and the

resulting situation after performing the action. Consider the translation for (9):

```
∃a∃b∃c∃k∃e(action(a,b,c)∧go(b,e)∧agent(b,e,robot)∧
kitchen(b,k)∧to(b,e,k))
```

The robot will react by inspecting the model generated for this representation and supporting background knowledge (including frame axioms). Typically, this supporting knowledge includes a notion of "actual world", the situation in which the robot is currently being part of. Hence, a model for (9) might look like:

```
D={d1,d2,d3,d4,d5,d6}

F(robot)=d1              F(go)={(d3,d5)}
F(world)={d2,d3,d4}      F(action)={(d2,d3,d4)}
F(actual)={d2}           F(agent)={(d3,d5,d1)}
F(to)={(d3,d5,d6)}       F(kitchen)={(d2,d6),(d3,d6),(d4,d6)}
```

In general, there might be several actions in a model: actions that have been executed in the past, actions that are possible to carry out in the current world, or actions to be performed in future situations. Models, however, directly lists which actions are relevant (those situated in the actual world) and which ones are not. Moreover, actions might be part of warnings (10). These are translated using universal quantification:

```
∃a∃s(stairs(a,s)∧
∀b∀c∀e(action(a,b,c)∧go(b,e)∧agent(b,e,robot)∧forw(b,e)
→fall(c,robot,s)))
```

If this information is supplied to the model builder, as well as with the translation for "Go forward!", the generated model will list the consequences of the action. If there are further axioms stating that falling down ought to be avoided, an inconsistent information state will result. This is the case for warnings, such as in (10). The robot might avoid this inconsistent state by disobeying the command "Go forward!".

Model building also helps with breaking down complex actions into atomic ones. Consider for instance commands involving universal quantification. Suppose our robot has a built-in vacuum cleaner and we tell it (10). Let's also supply some background knowledge that rooms can only be cleaned if they are dirty. Given a state of the house where there are three rooms based on the second floor, one of which is already clean, the model that will be generated will contain two actions, each describing the cleaning of a room.

3.3 Question Answering

The use of model building for question answering has been proposed in related work [11,2]. In this human-system dialogue scenario the user is allowed to

pose queries in natural language. The system answers these questions on the basis of the information provided by the model generated by the semantics of the question and background information (a database and possibly the ongoing dialogue between user and system). As shown in [2] with the CURT system, additional model checking is particular useful to present an answer. Furthermore, theorem proving is required for answer verification, because the answers available in the model do not always follow from the input theory (for instance in cases with disjunction). The CURT system is a nice example showing the use of a model builder, theorem prover and model checker in one architecture.

4 Using Model Builders in Practice

4.1 Experimental Setup

So far we've introduced model building, and presented three potential linguistic applications where the method could play an important role. But what are the limits of model building in practical applications anno 2003, employing analysis based on methods of formal semantics?

To find out, we undertook several experiments in an existing spoken dialogue system [4] which uses model building for semantic interpretation, using the two model builders MACE and PARADOX. The context in which the experiments were conducted is similar to that of controlling a robot, as described in the previous section. However, in this scenario we are able to control several domestic devices (lights, radios, fans) by spoken commands, such as:

Turn off a light.	(13)
Flip on every light.	(14)
Switch on all lights.	(15)
Turn on every light except the black light.	(16)
Switch the black light off. Switch it on.	(17)

The system managing the dialogue converts these utterances into Discourse Representation Structures, DRSs [8], resolving context-sensitive expressions such as pronouns, definite descriptions, and other presuppositional expressions. The DRSs obtained from the dialogue are then translated into first-order logic, and additional background knowledge (ontological information, theory of actions and possible worlds) stated as a first-order theory is calculated based on the content of the DRS. In addition, the current state of the devices in the environment (which devices are available, how they are distinguished from other devices, and whether they are on or off) is translated into first-order logic as well. We call this situational knowledge.

For each interpretation task, the translated DRS, the required background knowledge and the situational knowledge are given to a model builder, and at the same time the negation of the input formula is given to the theorem prover SPASS in order to find a counter-proof. For the purpose of this paper

we focus on the performances of the model builders MACE and PARADOX for this task. Both of these model builders search for models by iteration, but the performance of MACE can be considerably speed up by specifying an expected model size. PARADOX was used with parameter `--priority ConPredFun`. MACE was used with parameter `-k 200000` to allocate extra memory. We will refer to MACE* as the process of executing MACE with the additional parameter `-n X`, X specifying the size of the model, which was obtained empirically.

What is left to note is that the problems in this experiment are relatively hard for first-order inference engines, who are mostly tuned to mathematical problems. But linguistic problems, in particular those involving multi-sentence discourse, generate problems high in size (mostly due to background knowledge) and with a relatively high number of different predicates. Example (13), for instance, involves two 1-place predicates, forty-five 2-place predicates, and five 3-place predicates.

4.2 Results

The generated models were checked manually for correctness. The models contained information about the different devices and their status, the dialogue participants (user and system), and possible worlds, in this application used to model the flow of time. For example, the model generated for (13) contains nine entities: three devices, two participants, one event, and three possible worlds (one designating the current world, one designating the world in which the event of turning of the light takes place, and one designating the future world in which the effects of the turning on event hold).

Table 1
Timing results for model building for several example natural language commands.

Example	Number of Devices	Time (seconds) MACE	MACE*	PARADOX	Model Size
(13)	3	3.73	1.62	1.70	9
(14)	3	4.08	1.74	1.80	9
(15)	3	20.16	5.55	5.00	10
(16)	3	3.25	6.86	2.00	9
(17)	3	15.98	4.49	3.20	11

Table 1 shows performances of MACE and PARADOX for different linguistic phenomena, including simple indefinite noun phrases (13), quantification (14), plural descriptions (15), negation (16), and anaphora (17). The results show that the treatment of plurals is costly—this is not surprising, as extra axioms are triggered and a new entity in the model is required for modelling

sets. Similarly, even small discourses such as (17) require more computing—again because of the fact that the model increases, in this case the addition of two possible worlds.

Table 2
Timing results for model building with respect to increasing application domain sizes by adding devices. Results exceeding a time limit of 30 seconds are marked with "–". See Table 1 for further information on specifications.

Example	Number of Devices	Time (seconds) MACE	MACE*	PARADOX	Model Size
(13)	3	3.73	1.62	1.70	9
(13)	6	26.70	6.81	3.80	12
(13)	9	–	18.92	7.90	15
(17)	3	15.98	4.49	3.20	11
(17)	6	–	19.26	8.20	15
(17)	9	–	–	12.60	17

Table 1 also reveals that PARADOX outperforms MACE in most of the cases. The processing speech for finding a model for a particular problem can be highly reduced by supplying the expected domain size to the model builder, particularly in the case of MACE. These domain sizes are of course not known in advance while running practical applications, but maybe obtained using machine learning techniques. Although these results seem encouraging, they were obtained in a situation with only three devices. Table 2 shows some results by increasing the devices in the application domain, for instance by adding some lights, radios or fans. This results in growth of situational knowledge, with a strong negative impact on the performances of the model builders.

5 Conclusion

The results suggest that, on the one hand, it is possible to use first-order inference for natural language interpretation in practical applications with (very) small domains. On the other hand though, the approach doesn't seem to scale up very well. However, there are good reasons to believe that the situation can be improved.

First of all, the current approach doesn't work in an incremental way—all the information used for building a model is discarded as soon as the discourse is extended and reinterpreted. Different strategies have been proposed for incremental model generation in the context of natural language interpretation [1,9], but so far none have been implemented or used in practical systems.

Secondly, situational knowledge might be translated directly in the model representation rather than into first-order logic. A lot of the computation involved in the model construction experiments in this paper would be unnecessary if the model builder would permit pre-constructed models as initial values, rather than empty models. In the context of the described experiments, these pre-constructed models could contain all relevant devices and their status.

Thirdly, the size of the DRS can be constrained by suppressing information not relevant to the current topic of conversation. One idea is to use the "right frontier" of the unfolding discourse structure (the right-hand side of the discourse, represented as a tree [12]) and so minimise the amount of information given to the model builder. An alternative way to deal with this issue is assigning dynamic salience values to entities introduced into a discourse and formulating model generation rules in a resource sensitive way [10,5]. As a consequence, models are constructed using only entities whose salience values are high enough. Finally, the search space of the model builder can be reduced by using a sorted first-order logic.

Acknowledgements

I would like to thank Koen Claessen for assistence in running PARADOX, Bill McCune for help with MACE, and Patrick Blackburn, Ewan Klein, Alexander Koller, Malvina Nissim as well as three anonymous ICoS-reviewers for their comments on earlier versions of this article.

References

[1] Baumgartner, P. and M. Kühn, *Abducing coreference by model construction*, in: *Proceedings of the 1st Workshop on Inference in Computational Semantics (ICoS-1)*, Amsterdam, 1999, pp. 21–38.

[2] Blackburn, P. and J. Bos, *Representation and Inference for Natural Language. A First Course in Computational Semantics* (2003), draft available at `http://www.comsem.org`.

[3] Blackburn, P., J. Bos, M. Kohlhase and H. de Nivelle, *Inference and Computational Semantics*, in: H. Bunt, R. Muskens and E. Thijsse, editors, *Computing Meaning Vol.2*, Kluwer, 2001 pp. 11–28.

[4] Bos, J. and T. Oka, *An Inference-based Approach to Dialogue System Design*, in: *COLING 2002. Proceedings of the 19th International Conference on Computational Linguistics*, Taipei, Taiwan, 2002, pp. 113–119.

[5] Burchardt, A. and S. Walter, "BuGS, A Tableau Machine for Language Understanding," Master's thesis, Universität des Saarlandes (2001).

[6] Gardent, C. and K. Konrad, *Interpreting definites using model generation*, Journal of Language and Computation **1** (2000), pp. 193–209.

[7] Gardent, C. and B. Webber, *Towards the use of automated reasoning in discourse disambiguation*, Journal of Logic, Language and Information **10** (2001).

[8] Kamp, H. and U. Reyle, "From Discourse to Logic; An Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic and DRT," Kluwer, Dordrecht, 1993.

[9] Kohlhase, M., *Model generation for discourse representation theory*, in: *Proceedings of the 14th European Conference on Artificial Intelligence*, 2000, pp. 441–445.

[10] Kohlhase, M. and A. Koller, *Resource-Adaptive Model Generation as a Performance Model*, Logic Journal of the IGPL **11** (2003), pp. 435–456.

[11] Ramsay, A. and H. Seville, *Relevant Answers to WH-Questions*, in: P. Blackburn and M. Kohlhase, editors, *ICoS-3, Inference in Computational Semantics*, 2001, pp. 73–86.

[12] Webber, B. L., *Structure and Ostension in the Interpretation of discourse deixis*, Language and Cognitive Processes **6** (1991), pp. 107–135.